Chapter 4: Trees

Radix Search Trees

Lydia Sinapova, Simpson College

Radix Search Trees

- Radix Searching
- Digital Search Trees
- Radix Search Trees
- Multi-Way Radix Trees

Radix Searching

Idea: Examine the search keys one bit at a time

Advantages:

- reasonable worst-case performance
- easy way to handle variable length keys
- some savings in space by storing part of the key within the search structure
- competitive with both binary search trees and hashing

Radix Searching

- **Disadvantages:**
  - biased data can lead to degenerate trees with bad performance
  - for some methods use of space is inefficient
  - dependent on computer's architecture – difficult to do efficient implementations in some high-level languages

Radix Searching

- Methods
  - Digital Search Trees
  - Radix Search Tries
  - Multiway Radix Searching

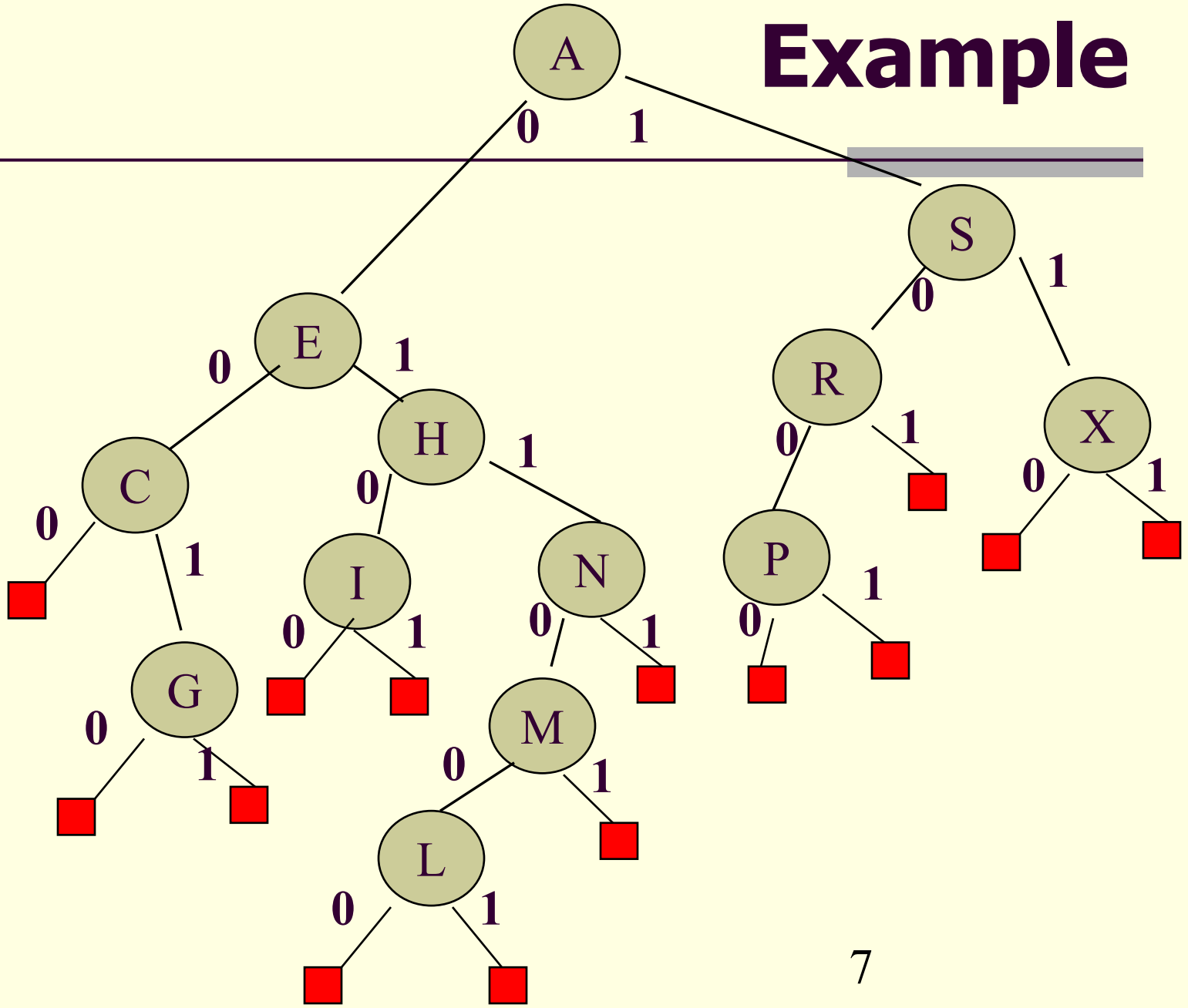
Digital Search Trees

- Similar to binary tree search
- Difference:

  Branch in the tree by comparing the key's bits, not the keys as a whole

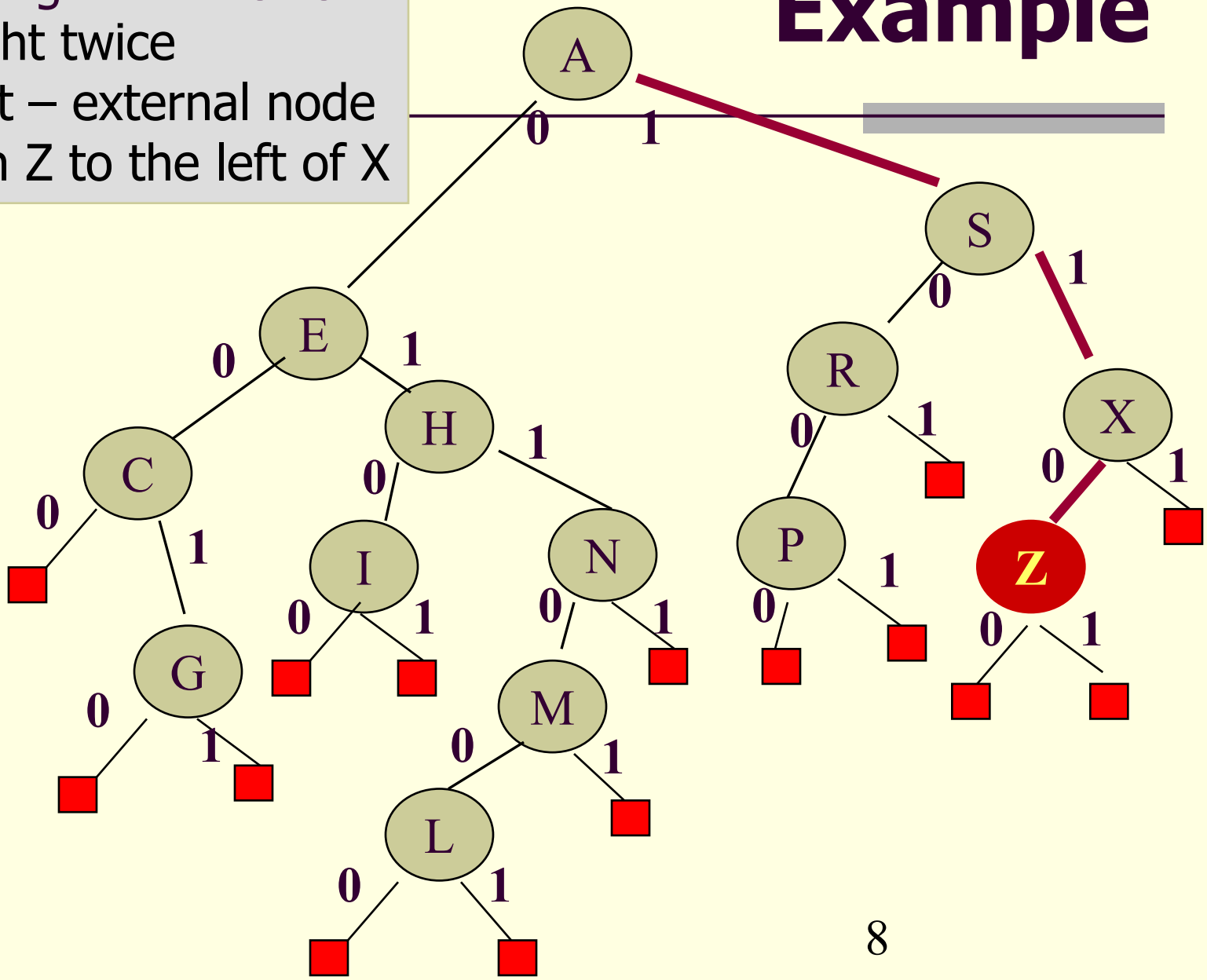
Example

A	00001
S	10011
E	00101
R	10010
C	00011
H	01000
I	01001
N	01110
G	00111
X	11000
M	01101
P	10000
L	01100



Example

inserting **Z = 11010**
go right twice
go left – external node
attach Z to the left of X



Digital Search Trees

Things to remember about digital search trees:

- **Equal keys are anathema** – must be kept in separate data structures, linked to the nodes.
- **Worst case – better than for binary search trees** – the length of the longest path is equal to the longest match in the leading bits between any two keys.

Digital Search Trees

- Search or insertion requires about log(N) comparisons on the average and b comparisons in the worst case in a tree built from N random b-bit keys.
- No path will ever be longer than the number of bits in the keys

Radix Search Trees

- If the keys are long digital search trees have low efficiency.
- **Radix search trees** : do not store keys in the tree at all, the keys are in the external nodes of the tree.
- Called **tries** (try-ee) from “retrieval”

Radix Search Trees

Two types of nodes

- **Internal:** contain only links to other nodes
- **External:** contain keys and no links

Radix Search Trees

To insert a key –

1. Go along the path described by the leading bit pattern of the key until an external node is reached.

2. If the external node is empty, store there the new key.

If the external node contains a key, replace it by an internal node linked to the new key and the old key. If the keys have several bits equal, more internal nodes are necessary.

NOTE: insertion does not depend on the order of the keys.

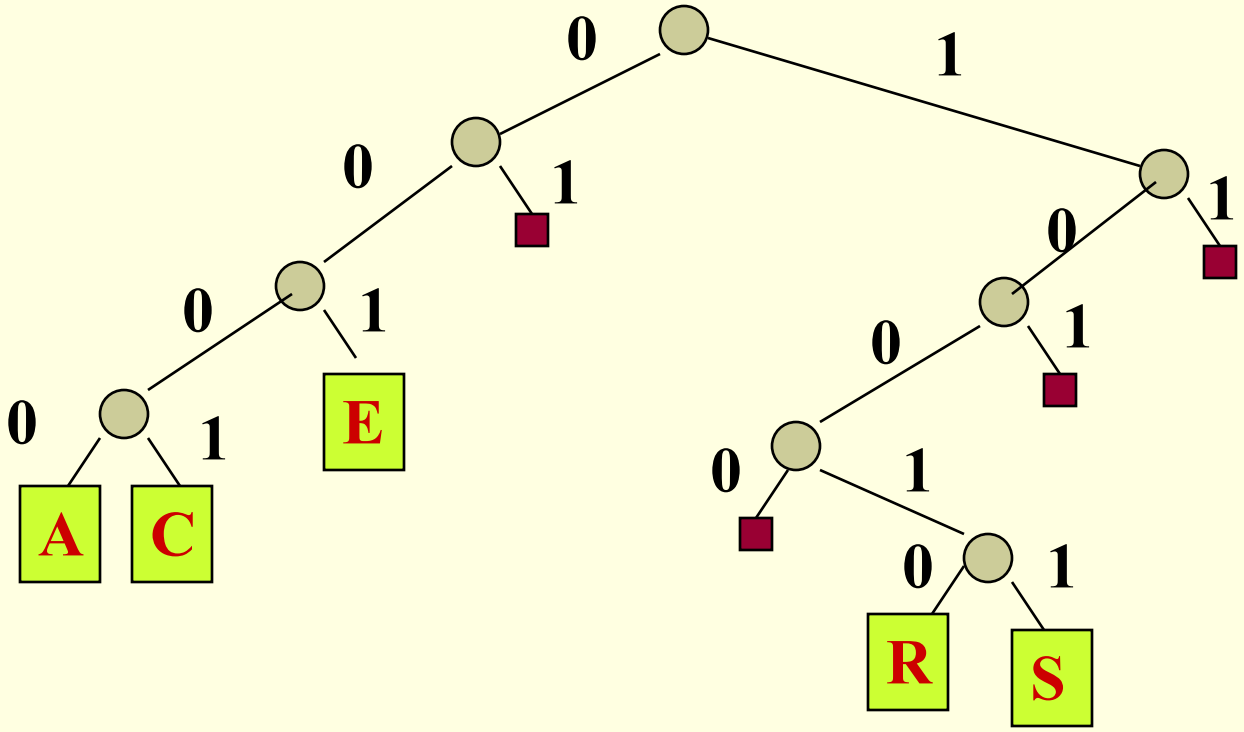
Radix Search Trees

To search for a key –

1. **Branch according to its bits,**
2. **Don’t compare it to anything, until we get to an external node.**
3. **One full key comparison there completes the search.**

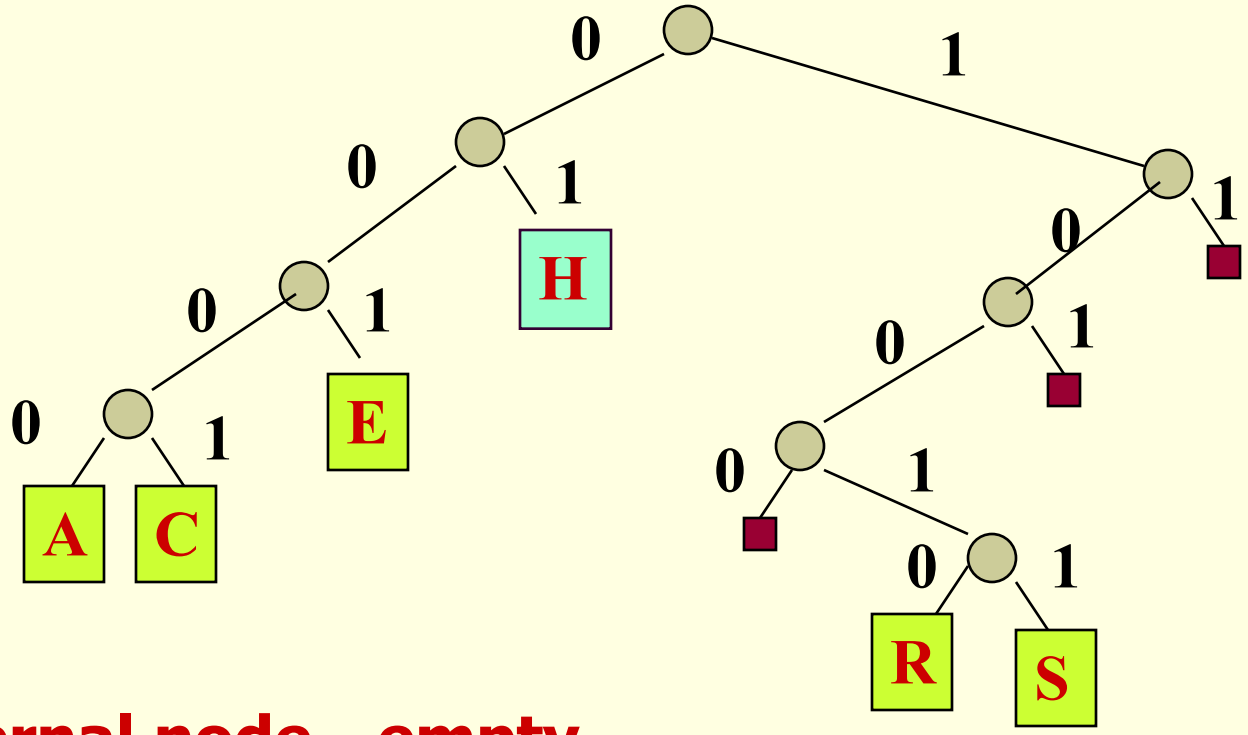
A 00001
S 10011
E 00101
R 10010
C 00011

Example



A 00001
S 10011
E 00101
R 10010
C 00011
H 01000

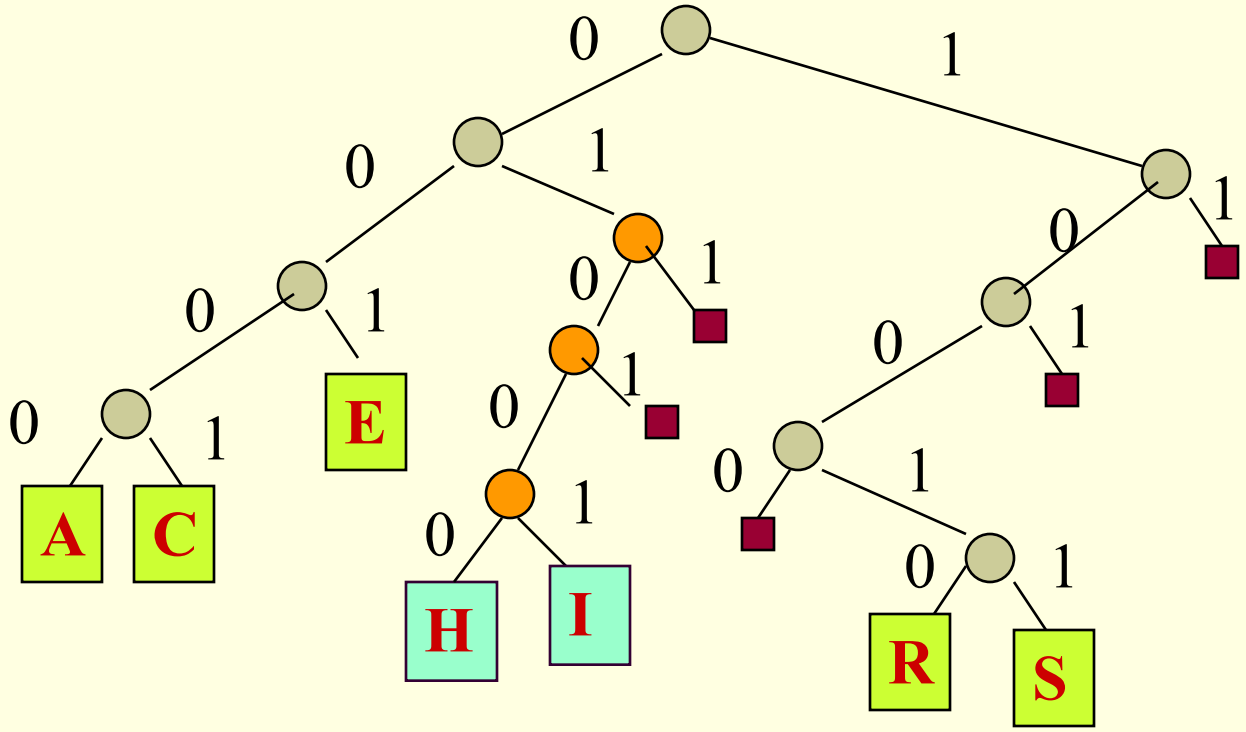
Example - insertion



External node - empty

A 00001
S 10011
E 00101
R 10010
C 00011
H 01000
I 01001

Example - insertion



External node - occupied

Radix Search Trees - summary

- **Program implementation -**
  - Necessity to maintain two types of nodes
  - Low-level implementation
- **Complexity:** about **logN** bit comparisons in average case and **b bit comparisons** in the worst case in a tree built from N random b-bit keys**.**

Annoying feature: One-way branching for keys with a large number of common leading bits :

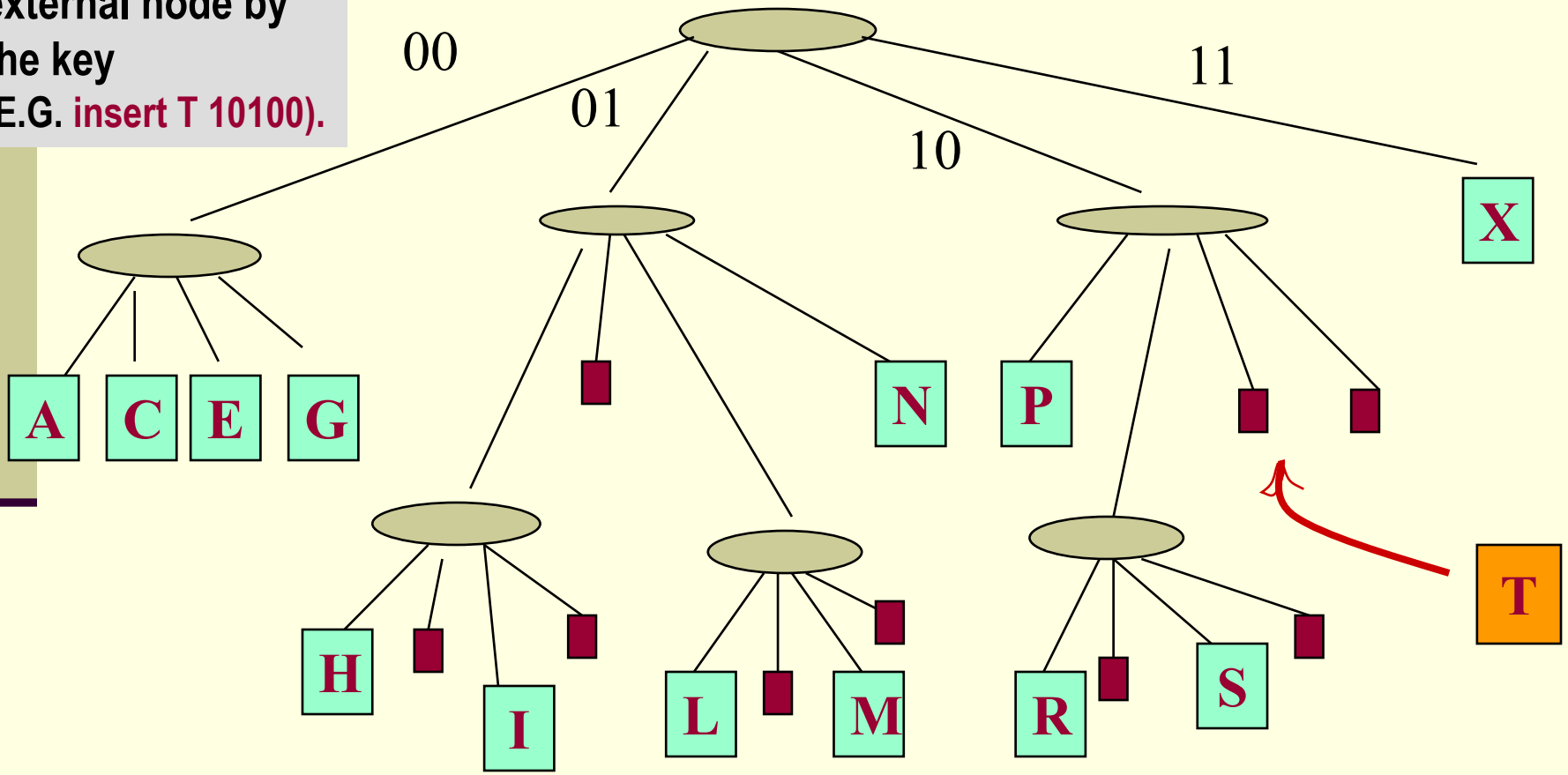
- The number of the nodes may exceed the number of the keys.
- On average – **N/ln2 = 1.44N nodes**

Multi-Way Radix Trees

- The height of the tree is limited by the number of the bits in the keys
- If we have larger keys – the height increases. One way to overcome this deficiency is using a multi-way radix tree searching.
- The branching is not according to 1 bit, but rather according to several bits (most often 2)
- If m bits are examined at a time – the search is speeded up by a factor of 2^m
- **Problem:** if m bits at a time, the nodes will have 2^m links, may result in considerable amount of wasted space due to unused links.

Multi-Way Radix Trees - example

Search – take left, right or middle links according to the first two bits.
Insert – replace external node by the key (E.G. insert T 10100).



Multi-Way Radix Trees

- Wasted space – due to the large number of unused links.
- Worse if M - the number of bits considered, gets higher.
- The running time: $\log_M N$ – very efficient.
- Hybrid method:
  Large M at the top,
  Small M at the bottom